Le système graphique ASH-PROLOG et son utilisation pour le prototypage rapide d'interfaces homme-machine

Alain Michard, Eric Monceyron

INRIA

UNITÉ DE RECHERCHE
INRIA-SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
BP 105
78153 Le Chesnay Cedex
France

Tél. (1) 39 63 55 11

Rapports Techniques

N° 76

LE SYSTÈME GRAPHIQUE ASH - PROLOG ET SON UTILISATION POUR LE PROTOTYPAGE RAPIDE D'INTERFACES HOMME-MACHINE

Alain MICHARD
Eric MONCEYRON

Octobre 1986

LE SYSTEME GRAPHIQUE ASH-PROLOG ET SON UTILISATION POUR LE PROTOTYPAGE RAPIDE D'INTERFACES HOMME-MACHINE.

THE ASH-PROLOG GRAPHICAL ENVIRONMENT AND THE FAST PROTOTYPING OF MAN-MACHINE INTERFACES.

A. MICHARD
E. MONCEYRON

INRIA - Centre de Sophia-Antipolis
06560 VALBONNE- France

RESUME

Pour répondre au besoin largement ressenti d'un outil de prototypage rapide des interfaces homme-machine modernes exploitant les écrans à haute résolution des stations de travail, nous avons interfacé un interpréteur Prolog largement répandu, et un environnement graphique de gestion de fenêtres. Nous montrons à travers un exemple détaillé que le résultat constitue un outil très agréable pour construire la maquette d'un logiciel interactif complexe, maquette aisément modifiable et donc facilitant la spécification de l'interface par itérations successives.

ABSTRACT

In order to facilitate the fast prototyping of man-machine interfaces on workstations with high-resolution bit-map screens, we have linked together a well-known Prolog interpreter and a portable screen and locator managing software. We try to show from an example that the resulting environment allows pleasant and comfortable development to the programmer who wishes to construct rapidly a modular declarative and executable specification of a user interface.

1. INTRODUCTION

La dissémination rapide des stations de travail et des micro-ordinateurs dotés de capacités d'affichage graphique de haute qualité provoque chez les utilisateurs professionnels ou privés une élévation sensible de leur degré d'exigence vis à vis de la qualité de l'interface homme-machine des logiciels qu'ils utilisent.

Dans le domaine des applications grand public les modes de dialogue basés sur la désignation de représentations graphiques des objets manipulés, tendent à supplanter les langages de commande à base de touches codées et les messages littéraux plus ou moins abscons qui caractérisaient trop souvent les logiciels concus à l'époque des terminaux télescripteurs. En programmation professionnelle, les privilegiés (pour encore peu de temps !) ayant gouté aux environnements multi-fenêtres sur grand écran des postes de travail modernes, manifestent quelques réticences à devoir parfois revenir à leur "vieux" terminaux limités à 24 lignes et 80 colonnes.

L'amélioration considérable de la qualité ergonomique moyenne des interfaces utilisateurs, rend de plus en plus inévitable de prévoir dans le cycle de conception des logiciels une phase de *prototypage* de cette interface : d'une part il devient inacceptable de se tromper lourdement dans ce domaine (ailleurs aussi sans doute !) sous peine de constater des attitudes de refus et de rejet du client utilisateur, et d'autre part cette interface tend à représenter dans les logiciels actuels une part de plus en plus importante de l'effort de développement, qu'il soit mesuré en hommes x mois ou en lignes de code source.

Comme tout prototypage logiciel, le prototypage d'interfaces implique une conception descendante par étapes (la spécification se faisant de plus en plus précise), et une évaluation régulière des choix faits à l'aide de critères aussi objectifs et fidèles que possibles. Plus spécifiquement le prototypage à vocation ergonomique nécessite une ou plusieures évaluations expérimentales de l'interface réalisée, au cours desquelles un échantillon représentatif de la population des futurs utilisateurs devra réaliser une ou plusieurs tâches tests. Le concepteur pourra selon son degré d'exigence se contenter de critères de satisfaction subjectifs, (opinion du groupe témoin) faciles à recueillir mais peu fiables, ou tenter de recourir à tout ou partie de la panoplie des indices comportementaux qu'utilisent les ergonomes : enchainements non optimaux, erreurs, catachrèses, ajournement de sous-tâches, etc.. Nous ne decrirons pas ici la méthodologie de l'évaluation ergonomique, mais il importe de noter que lors de ce cycle d'évaluation, l'interface réalisée peut éventuellement être totalement invalidée et refondue. Ces profondes remises en cause des choix antérieurs ne sont pratiquement possibles que si l'environnement de prototypage les facilite. En particulier nous attendons de cet environnement qu'il autorise:

- une grande rapidité de développement;
- les modifications faciles d'une version;
- la réutilisation de "parties de code"[1] d'une version à l'autre;

Pour notre part nous ajouterons comme exigences que cet environnement soit basé sur un langage de spécification

- *déclaratif* (lisibilité du code, facilité des modifications);
- *exécutable* : il faut pouvoir *interpréter* les spécifications et obtenir un résultat immédiat;
- *universel* : nous ne voulons pas d'un formalisme dédié et donc limité à la description des interfaces, mais un langage permettant le prototypage de l'ensemble du logiciel cible.

---

[1] Nous n'utilisons pas ici le mot "module" qui a en programmation un sens technique plus précis que ce que nous souhaitons désigner.

Divers outils ont été proposés qui satisfont -plus ou moins- à ces exigences, certains totalement ad-hoc ([Clark,81],[Hayes,85],[Wasserman,85]), d'autres plus généraux. Parmi ces derniers le langage Prolog nous semble présenter de grandes qualités en tant que langage de prototypage [Rueher,85] à la condition de le doter des capacités de manipulation graphique qui lui manque. Avoir avoir décrit une implémentation d'un Prolog graphique réalisée à partir de composants existants, nous montrerons comment ce langage permet de facon agréable et -croyons nous- efficace de spécifier et d'évaluer des interfaces homme-machine complexes.

2. L' OUTIL DE BASE : ASH_PROLOG.

2.1. ASH : le gestionnaire d'écran.

ASH ("A Screen Handler"), est un gestionnaire de fenêtres écrit en C, portable sur de nombreux postes de travail Unix (Appollo, Sun, SPS7, SPS9..). Il fait partie du "Brown Workstation Environment" (BWE) développé à L'Université Brown [Pato,84], pour répondre au besoin d'un environnement graphique *unique* disponible sur les postes de travail les plus répandus. ASH est par ailleurs utilisé dans le système Le_Lisp [Chailloux,85] pour donner à ce système lisp les capacités graphiques qui lui manquaient.

Il offre au programmeur C un ensemble de types prédéfinis -en particulier le type "fenêtre" < ASH_WINDOW> - et une bibliothèque de fonctions permettant de créer, déplacer, cacher, etc.. des fenêtres, et d'y effectuer toutes les opérations graphiques classiques. L'ensemble des fenêtres est géré selon un modèle d'arbre, chaque nouvelle fenêtre étant définie par rapport à une fenêtre mère. Les attributs de visualisation d'un noeud sont en règle générale rémanents sur le sous-arbre.

BWE propose également dans la bibliothèque "APIO" tout ce qui est nécessaire pour récupérer les entrées utilisateur --> système dans une pile unique d'évènements. Une fonction (APIOget) permet d'examiner cette pile et fournit pour chaque évènement son type (clavier ou souris) et les valeurs des paramètres utiles: coordonnées X Y ou bouton pressé pour la souris, caractère pour le clavier.

Enfin la bibliothèque VT (Virtual Terminal) permet d'utiliser toute fenêtre ASH comme un terminal virtuel.

2.2. Prolog.

Nous ne présenterons pas ici le langage Prolog, largement répandu depuis quelques années et pour lequel existent quelques bons manuels ([Clocksin,81], [Bratko,86],[Kluzniak,85]). Nous utilisons l'interprète écrit en C à l'Université d'Edimbourg, en raison de son efficacité, de sa portabilité sur toute machine Unix, et du fait que sa syntaxe se soit imposée comme une norme de fait (bientôt de droit ?) internationale. De plus le fait que cet interprète Prolog et ASH soient écrits dans le même langage facilite évidemment l'interfacage des deux.

2.3. ASH_PROLOG : construction et fonctionnalités.

Pour permettre l'accés depuis Prolog aux fonctionnalités offertes par ASH, nous introduisons une centaine de nouveaux prédicats prédéfinis, qui pour la plupart ont pour effet de bord de déclencher directement une fonction ASH. Le passage de paramètres entre ASH et Prolog est assuré par une structure de données particulière[2], et par une fonction rajoutée au code de l'interprète pour récupérer les valeurs des variables instanciées lors de l'évaluation d'un prédicat graphique.

---

[2] Selon une suggestion de D. Clément, Projet CROAP, INRIA, que nous remercions ici de son aide précieuse.

On trouvera en Annexe la liste commentée de tous les prédicats disponibles, organisée selon les rubriques suivantes:

-a- des prédicats d'initialisation de l'environnement graphique;

-b- des prédicats de création et de manipulation de fenêtres, permettant la création, la selection, la gestion de la visibilité des fenêtres, l'obtention d'informations sur leur positionnement et leur visibilité réelle à un instant donné, le déplacement soit par programme soit par l'utilisateur de fenêtres, etc..

-c- des prédicats graphiques permettant le tracé de points, vecteurs, surfaces pleines, cercles, ellipses, etc..

-d- des prédicats permettant d'écrire à l'emplacement choisi des caractères ou chaines de caractères, dans la police choisie, avec des possibilités de mise en page au moins équivalentes à celles disponibles sur un éditeur de texte classique;

-e- des prédicats de création et de gestion des zones sensibles aux désignations, ces zones pouvant être activées et désactivées à volonté;

-f- des prédicats permettant de récupérer les "évènements utilisateur", désignations ou clavier;

-g- divers prédicats n'appartenant pas à ces catégories, et pourtant bien utiles pour certains d'entre eux !! [3]

De plus pour éviter d'avoir à gérer des boucles d'attente en Prolog, un certain nombre de prédicats prédéfinis plus complexes ont été introduits : c'est le cas par exemple de

apiogetstring(X).

permet de récupérer directement dans X toute une chaine de caractères entrée au clavier, et qui assure l'écho des caractères dans la fenêtre courante, l'édition interactive de la chaine par le caractère <retour>, la gestion du curseur et le déroulement horizontal automatique de la ligne en cours de saisie (tout ceci étant facilité par l'utilisation des fonctions de la bibliothèque VT mentionnée ci-dessus);

apiogetclick(X,Y,S).

renvoie les coordonnées X et Y de la souris au moment d'une pression sur le bouton S;

ashpopup(Z).

renvoie le nom de la zone sensible Z désignée au moment du *relachement* du bouton de la souris.

move_window(Nx,Ny).

permet à *l'utilisateur* de déplacer une fenêtre, en maintenant le bouton de la souris appuyé. Pendant le déplacement, un cadre de même dimensions que la fenêtre matérialise la position virtuelle obtenue.

Le lecteur habitué à l'écriture d'interfaces homme-machine aura sans doute remarqué que ces fonctionalités restent relativement de bas niveau : on n'y voit pas apparaitre en tant que telles les fonctions de définition de menus (dynamiques ou statiques,en ligne, colonne ou autre..), de zones de messages, etc.. Nous allons voir en effet que Prolog se prête parfaitement à la définition de ces notions, et c'est volontairement pour donner à l'utilisateur un contrôle complet de ces définitions que nous ne les avons pas introduites sous forme de prédicats prédéfinis et donc figés.

---

[3] Cette liste correspond aux possibilités de la version 2.2 du logiciel (Septembre 86). La version suivante sera enrichie de nouvelles fonctionalités de plus haut niveau, définies en Prolog.

2.3.1. Remarque sur le passage de paramètres.

Le mécanisme de passage de paramètres entre Prolog et ASH est limité à des entiers. Or de nombreuses fonctions de ASH utilisent des noms symboliques en particulier pour préciser différents attributs graphiques des lignes, bordures, fenêtres, zones sensibles, etc.. De plus beaucoup des fonctions rendent un pointeur sur un objet typé qui peut être affecté à une variable qui servira à manipuler le-dit objet. La création d'une fenêtre en C se fera selon le schéma:

```
ASH_WINDOW ma_fenetre;          /* déclaration d'une variable du type "fenêtre" */

...

ma_fenetre = ASHcreate(0,250,0,125,350,0,ASH_BORDER_THIN, ASH_WINDOW_INVISIBLE);
                    /* la fonction ASHcreate rend un pointeur sur un objet
                       du type ASH_WINDOW; l'affectation permettra de manipuler
                       cet objet (le selectionner..) en le désignant par le
                       nom de la variable "ma_fenetre" */
```

Il était nécessaire de fournir au programmeur Prolog la possibilité d'une part de désigner les objets créés par ASH par des atomes, et d'autre part de lui permettre d'utiliser les noms symboliques désignant les valeurs possibles des paramétres des fonctions graphiques. Nous utilisons pour ce faire la base de faits interne à Prolog: d'une part à l'initialisation du système, tout un ensemble de prédicats sont "assertés" donnant les correspondances entre les noms symboliques des valeurs des paramétres et leur valeur numérique qui sera passée à ASH, d'autre part lors de chaque création d'un objet graphique (fenêtre, zone sensible,..) un prédicat est automatiquement créé dans la base de faits pour mémoriser la relation entre son nom symbolique et son numéro pour ASH.

3. LA CONSTRUCTION D'INTERFACES HOMME-MACHINE AVEC ASH-PROLOG

Essayons tout d'abord de donner une définition précise de la notion d'interface homme-machine (IHM).

Une IHM est une procédure de dialogue, dont la définition comporte:[4]

-a- des objets graphiques élémentaires pré-définis : caractères, point, vecteur, curseurs,icones,boites,cercle,ellipse, etc.;

-b- un catalogue d'événements "utilisateur" détectables: frappe d'un caractère, déplacement de la souris, appui sur un bouton-souris ou sur une touche fonction;

-c- des structures graphiques, définissant des unités affichables *élémentaires* en ce sens qu'elles ne sont pas dissociables. Chacune de ces structures graphiques peut être décrite comme une hiérarchie (arbre) de composants, chaque noeud de cette hiérarchie étant suceptible d'être étiqueté par des attributs de visualisation (visibilité, inversion video, clignotement, type de ligne, police de caractères,...) valides sur tout le sous-arbre correspondant. Les relations topologiques entre les composants d'une même structure sont figées. Par exemple un menu vertical est composé d'un bandeau-titre, d'un ensemble de boutons eux-mêmes composés d'une boite,d'une étiquette, d'une bordure,etc.. et le positionnement relatif de ces éléments pour un certain menu, ne changera jamais au cours de l'interaction.

-d- des *écrans* formés à l'aide des structures précédentes et d'un ensemble de relations géométriques permettant de les positionner les unes par rapport aux autres à un certain instant t de l'interaction.

---

[4] Nous n'envisageons ici que les IHM "classiques"mettant en jeu écran, clavier et dispositif de désignation (souris). Nous n'avons pas réfléchi aux extensions du modèle nécessaires pour prendre en compte le canal vocal (synthèse et reconnaissance).

-e- des règles d'enchainement des écrans ainsi définis, definissant les transitions entre les états t, $t+1$ .. t_i, en fonction des événements survenant au cours de l'interaction. Les événements peuvent être soit "utilisateur" (clavier, souris) soit "machine" (fin d'un calcul, arrivée d'un message sur le réseau..). En général, tout événement utilisateur implique une transition et un changement de l'état de l'écran, ne serait-ce que l'écho d'un caractère par exemple, alors que les événements machine ne provoquent de transition que sous certaines conditions. Les règles de transition ne sont donc pas indépendantes du contexte, c'est à dire des événememts qui ont précédés leur activation.

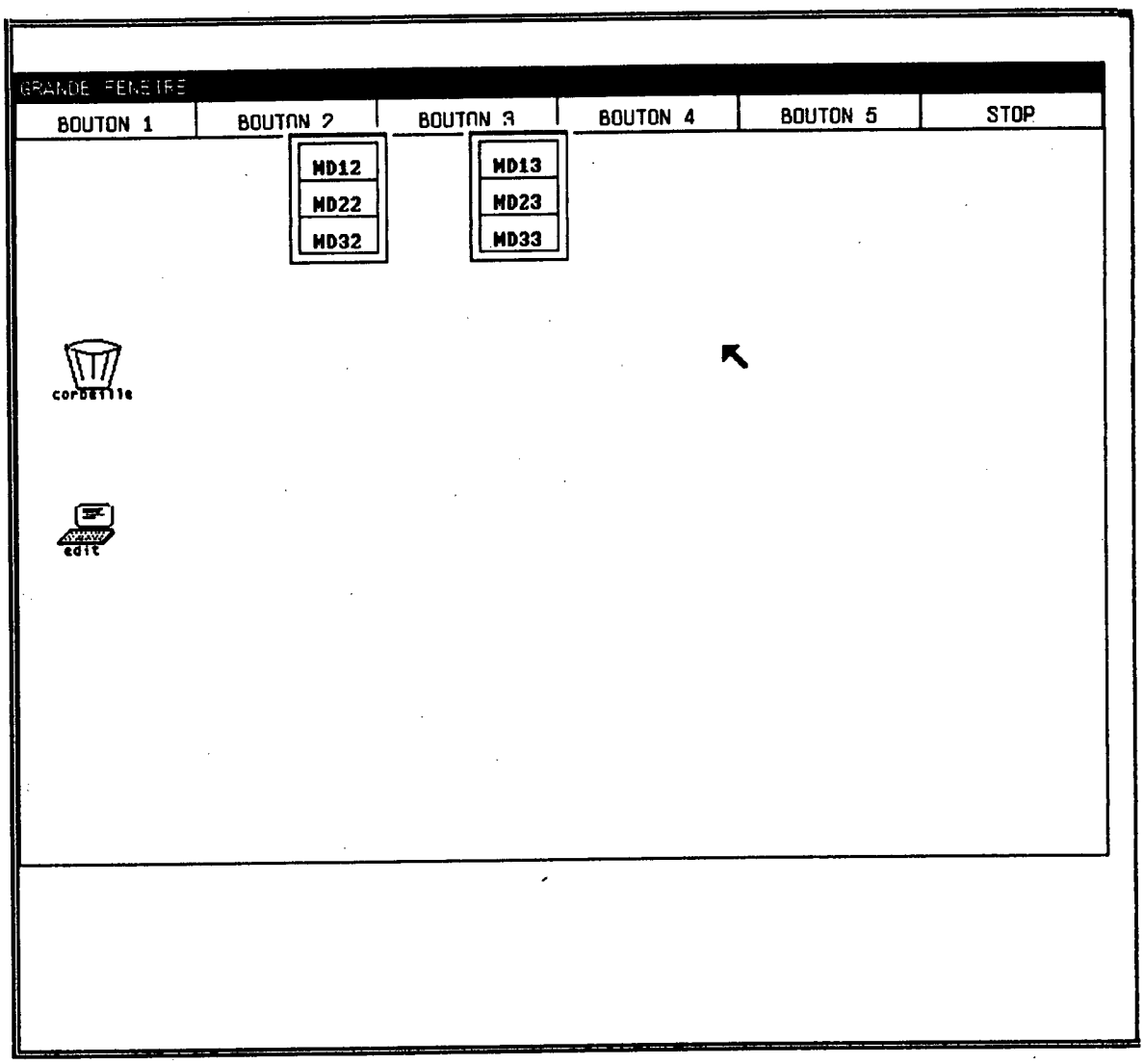
Nous allons montrer à travers un exemple simple mais détaillé que ASH-PROLOG constitue un formalisme commode pour spécifier une IHM particulière et pour la valider.

3.1. Un écran à menus multiples.

La figure 1 montre un exemple d'écran asses simple composé d'un menu horisontal supérieur et d'une sone centrale comportant plusieurs icones désignables. A chaque bouton (1 à 5) du menu superieur correspond un menu vertical dynamique. Le sixieme bouton provoque l'arrêt du programme Prolog. Les icones peuvent être déplacées à l'aide du bouton 1 de la souris, ou être activées par les deux autres boutons. Pour limiter les dimensions de cette exemple, nous nous contenterons de faire correspondre à chaque désignation sur un menu dynamique l'affichage d'un message dans une sone de dialogue.

Figure 1: I.H.M. à menus multiples et icones.

En fonctionnement normal les deux menus dynamiques n'apparaissent jamais simultanément.



3.1.1. Les composants graphiques.

Définissons notre fenêtre de base:

```
grande_fenetre_dfn :-
        ashinq_top(Z),ashselect(Z),
        makewindow(grande_fenetre,0,550,0,500,720,0,ash_border_window,ash_window_visible),
        ashset_window_name('GRANDE FENETRE').
```

Le prédicat *ashinq_top* nous rend le nom de la fenêtre "racine" désignant soit le plein écran, soit une fenêtre graphique de notre environnement de travail depuis laquelle le programme sera lancé. Nous y créons *grande_fenêtre* dont le coin inférieur gauche est au coordonnées X= 0, Y= 550, dans la fenêtre racine, et dont les dimensions sont 720 sur 500 (en pixels). [5] Nous lui affectons l'étiquette "GRANDE FENETRE", qui apparaitra dans son bandeau supérieur.

Nous pouvons maintenant définir notre menu horizontal:

```
menu_de_base_dfn :-
        ashselect(grande_fenetre),
        makewindow(menu_de_base,0,24,0,24,720,0,ash_border_none,
                ash_window_invisible or ash_window_hit_use),
        ashcenter_text('BOUTON 1',0,24,120,0),
        ashcenter_text('BOUTON 2',120,24,240,0),
        ashcenter_text('BOUTON 3',240,24,360,0),
        ashcenter_text('BOUTON 4',360,24,480,0),
        ashcenter_text('BOUTON 5',480,24,600,0),
        ashcenter_text('STOP',600,24,720,0),
        makesensitive_area(btn_1,0,24,120,0,ash_sense_flip),
        makesensitive_area(btn_2,121,24,240,0,ash_sense_flip),
        makesensitive_area(btn_3,241,24,360,0,ash_sense_flip),
        makesensitive_area(btn_4,361,24,480,0,ash_sense_flip),
        makesensitive_area(btn_5,481,24,600,0,ash_sense_flip),
        makesensitive_area(btn_6,601,24,720,0,ash_sense_flip),
        ashbox(0,0,120,24),
        ashbox(120,0,240,24),
        ashbox(240,0,360,24),
        ashbox(360,0,480,24),
        ashbox(480,0,600,24),
        ashbox(600,0,720,24).
```

Aprés avoir selectionné notre grande fenêtre, nous créons la fenêtre *menu_de_base* qui servira de support à notre menu horizontal. Nous centrons le nom de nos six boutons dans six boites virtuelles d'égales dimensions, que nous transformons ensuite en six zones sensibles aux désignations: lorsque l'utilisateur cliquera sur l'une de ces zones, un prédicat nous permettra de récupérer directement son nom symbolique (*btn_5* par exemple). La présence du paramètre *ash_sense_flip* fera que ces zones passeront en inverse video chaque fois que l'utilisateur y positionnera son curseur. Nous décidons ensuite, (choix esthétique donc facultatif) d'encadrer ces six boutons d'un liseré noir *(ashbox)*.

Pas très jolie, et un brin fastidieuse, cette définition des 6 boutons de notre menu, n'est-ce pas? Nous allons voir à l'occasion de la définition des menus verticaux qu'une solution plus élégante aurait pu être mise en oeuvre: en effet, les menus "en bande", verticale ou horizontale, de longueur variable, sont très souvent utilisés dans les IHM qui nous intéressent. Dans un souci d'économie et de concision, nous allons nous donner une fonction capable de générer un menu

---

[5] Le systéme de coordonnées écran a son origine 0,0 au coin supérieur gauche. L'abcisse X est horizontale.

vertical quelconque à partir d'une liste d'étiquettes et d'une liste de boutons:

```
define_vertic_menu(Name,Label_list,Button_list,X,Y) :-
        longest_word(Label_list,W),              /*le mot le plus long*/
        ashinq_text_extent(W,Xmax,Ymax),         /*dimensions de ce mot*/
        length(Label_list,N),                    /*combien de mots? */
        Xextent is Xmax + 20,
        Yextent is (Ymax + 8)*N,                 /*dimensions du menu */
        Ydim is Ymax + 8,
        Yoffset is Y + Yextent,
        makewindow(Name,X,Yoffset,0,Yextent,Xextent,0,ash_border_sensitive,
                ash_window_invisible or ash_window_hit_use),
        ashselect(Name),
        ashfont('screen.b.12'),
        draw_boxes(Label_list,Xextent,Ymax,0,rect_limits),
        make_sensitive_areas(Button_list,Xextent,Ydim,0),
        ashfont('screen.r.12').

draw_boxes([], _, _, _, _).
draw_boxes([H|T],Xextent,Ymax,Yoff,no_limits) :-
        Yoffset is Yoff + Ymax + 8,
        Ychar is Yoff + Ymax + 4,
        ashtext(10,Ychar,H),
        draw_boxes(T,Xextent,Ymax,Yoffset,no_limits).
draw_boxes([H|T],Xextent,Ymax,Yoff,rect_limits) :-
        Yoffset is Yoff + Ymax + 8,
        Ychar is Yoff + Ymax + 4,
        ashtext(10,Ychar,H),
        ashbox(0,Yoff,Xextent,Yoffset),
        draw_boxes(T,Xextent,Ymax,Yoffset,rect_limits).


make_sensitive_areas([], _, _, _).
make_sensitive_areas([H|T],Xdim,Ydim,Yoff) :-
        Ymax is Yoff + Ydim,
        makesensitive_area(H,0,Ymax,Xdim,Yoff,ash_sense_flip),
        make_sensitive_areas(T,Xdim,Ydim,Ymax).


longest_word([],[]).
longest_word(L,H) :- insort(L,[H|T]). /* solution de facilite! */

insort([],[]).
insort([X|L],M) :- insort(L,N), insortx(X,N,M).
insortx(X,[A|L],[A|M]) :- order(A,X), !, insortx(X,L,M).
insortx(X,L,[X|L]).        /* pris tel quel dans CM */

order(A,B) :- name(A,La),name(B,Lb),
              length(La,Na),length(Lb,Nb), Na > Nb
```

Un menu vertical de nom *name* , dont la liste des étiquettes est dans *Label_list* et dont la liste des noms de zones sensibles est *Button_list* , postionné à la création en X,Y dans sa fenêtre mère, se définit par la procedure suivante:

Les dimensions *Xmax,Ymax* du mot le plus long *W* de *Label_list,* vont nous permettre de déterminer la largeur *Xextent* du menu. Sa hauteur *Yextent* dépend de la hauteur d'un mot

(pour tenir compte des polices de caractères) et du nombre d'items. On fabrique la fenêtre correspondante et l'on sélectionne sa police préférée (cela aurait pu être passé en paramétre?). Il ne reste plus qu'à positionner les étiquettes et les cadres *(draw_bozes)* ,et à définir les sones sensibles *(make_sensitive_areas)*.

draw_bozes est une fonction récursive qui va inscrire des étiquettes (avec ou sans encadrement) jusqu'à épuisement de la liste *Label_list*. *"make_sensitive_areas"* fonctionne sur le même principe, sur la liste *Button_list*. (Il est indispensable que ces deux listes aient le même nombre d'éléments: il n'y a aucun contrôle exercé par ces fonctions).

longest_word est l'utilitaire qui rend l'atome le plus long d'une liste L.

Nous pouvons maintenant très facilement définir nos menus verticaux:

```
menu_dynam_1_dfn :-
        ashselect(grande_fenetre),
        define_vertic_menu(menu_dynam_1,
                        ['MD1','MD2','MD3'],          /* etiquettes        */
                        [md11_btn, md21_btn, md31_btn],   /* boutons du menu1  */
                        0,100),
        ashselect(menu_dynam_1),
        ashinq_size(ash_size_window_full,Lx,By,Rx,Ty),   /* quelle taille ??  */
        Xoffset is (120 - (Rx - Lx))/2,
        Yoffset is (24 + (By - Ty) +1),                  /* on le met en place */
        ashview(Xoffset,Yoffset,Lx,By,Rx,Ty).            /* centré sur la case 1 */
```

... et ainsi de suite pour les quatre autres. Le seul commentaire à faire est que l'on a choisi de cadrer les menus verticaux à droite par rapport aux boutons du menu horizontal.

Il nous reste a positionner les icones (fabriquées antérieurement avec votre éditeur graphique préféré). Dans la version actuelle (2.2) les icones font impérativement 64 x 64.[6]

```
icons_dfn :-
        ashselect(grande_fenetre),
        makewindow(edit_icon_wn,20,108,0,64,64,0,ash_border_none,
                ash_window_hit_use),
        ashload_icon(0,64,edit_icon),
        makesensitive_area(edit_icon_btn,0,64,64,0,ash_sense_no_change),
                                /* seconde icone     */
        ashselect(grande_fenetre),
        makewindow(trash_icon_wn,20,196,0,64,64,0,ash_border_none,
                ash_window_hit_use),
        ashload_icon(0,64,trash_icon),
        makesensitive_area(trash_icon_btn,0,64,64,0,ash_sense_no_change).
```

Les trois opérations sont dans l'ordre: création d'une fenêtre de 64 x 64, chargement du bit-map de l'icone, et création de la zone sensible correspondante.

Enfin définissons une zone de message au milieu de la grande fenêtre:

```
prompt_zone_dfn :- ashselect(grande_fenetre),
        makewindow(prompt_area,50,300,0,200,600,0,
                        ash_border_thin, ash_window_invisible).
```

[6] Dans la version 2.2 cette fonction n'est disponible que sur SUN.

3.1.2. Logique des enchainements.

A partir des éléments dont nous disposons maintenant, définissons la logique de fonctionnement de notre IHM: à chaque zone sensible de l'écran (et éventuellement pour chaque bouton de la souris), nous allons faire correspondre une action :

```
exec(trash_icon_btn,1) :- apiogetclick(_, _,1),
            move_window(_, _),!.        /*deplacement de l icone*/

exec(trash_icon_btn, _) :- prompt('MANIPULATION DE LA CORBEILLE').

exec(edit_icon_btn,1) :-  apiogetclick(_, _,1),
            move_window(_, _),!.

exec(edit_icon_btn, _) :- prompt('APPEL DE EDITEUR').

exec(btn_1, _) :- ashselect(menu_dynam_1),
            ashvisible(1),
            ashpop,
            apiogetclick(_, _,S),
            ashpopup(Z),
            ashvisible(0),
            ashpush,
            exec(Z,S).
```

Le prédicat *apiogetclick(X,Y,S)* attend un click souris de l'utilisateur. Il rend les coordonnées *X* et *Y* au moment du click, et le numero *S* du bouton de la souris en cas d'utilisation d'une souris qui en comporte plusieurs. *wait_loop* est un predicat d'attente d'une désignation sur une zone sensible (voir ci-dessous), *move_window(X,Y)* permet à l'utilisateur de déplacer une fenêtre en maintenant le bouton de la souris appuyé et rend les coordonnées à la fin du déplacement. *ashpush* et *ashpop* permettent respectivement de "pousser" une fenêtre "au fond" de l'écran (sous les autres), et de la rappeler "au dessus" de l'écran.

De même on associe à chaque bouton d'un menu dynamique une sémantique. Par exemple :

```
exec(md21_btn, _) :- prompt('BOUTON 2 du MENU 1').
```

Le dernier élément manquant à notre IHM est le prédicat *prompt* que nous avons largement utilisé dans la description des actions, et qui n'offre aucune difficulté particulière:

```
prompt(S) :-   ashselect(prompt_area),
            ashvisible(1),
            ashpop,
            ashfont('screen.b.14'),
            ashtext(10,30,S),
            ashfont('serif.r.10'),
            ashtext(20,50,'Cliques pour effacer ce message '),
            apiogetclick(_, _, _),
            ashvisible(0),
            ashpush,
            ashselect(menu_de_base).
```

C'est fini !! Pour tester notre exemple, il nous suffit de définir un prédicat *run* :

```
run :- ashinit,
       grande_fenetre_dfn,
       menu_de_base_dfn,
       icons_dfn,
       menu_dynam_1_dfn,
       menu_dynam_2_dfn,
       menu_dynam_3_dfn,
       prompt_zone_dfn,
       ashselect(menu_de_base),
wait_loop.
```

la boucle d'attente des désignations étant définie elle-même ainsi:

```
wait_loop :- repeat, apioflush,
     ashinq_sensitive(X,S),exec(X,S), fail.
```

On remarquera que l'emploi d'une boucle *repeat ... fail* évite la saturation de la mémoire de Prolog que provoqueraient les empilements d'environnements dus à l'emploi d'une fonction récursive.

Bien évidemment nous n'avons pas dans ce court exemple utilisé toutes les fonctionnalités de ASH-PROLOG.[7] Nous allons en utiliser quelques autres dans le second exemple.

3.2. Affichage d'un fichier de texte dans une fenêtre.

Il s'agit là d'une fonction souvent utilisée dans les IHM, et qui illustre bien l'utilisation d'un terminal émulé dans une fenêtre. Le programme est le suivant:

[7] Notre exemple, volontairement très simple, devrait pour être complet prévoir un certain nombre de coupures ("cut") pour éviter, lors de la mise au point en particulier, des tentatives de backtracking inutiles ou même néfastes: en règle générale et sauf exception volontaire, il vaut mieux interdire tout backtrack sur des prédicats graphiques.

```
init :- ashinit,
        makewindow(base_wn,0,500,0,400,600,0,
              ash_border_tab,ash_window_hit_use).

ecrire(File) :- ashselect(base_wn),
              vt_open,
              look(File).


look(F) :- seeing(I),
          see(F),
           looking,
          close(F),
          see(I).

looking :- repeat,
            readstring(S),
            vt_display(S),
             vt_display('
'),
            test_ligne(S),
          !.

test_ligne(end_of_file).
test_ligne(_) :- fail.
```

Aprés création de la fenêtre, initialisation du terminal virtuel, et ouverture du fichier, les lignes sont lues une par une et affichées immédiatement, et ce jusqu'à épuisement du fichier. L'atome *end_of_file* et le prédicat *readstring(S).* sont prédéfinis dans notre bibliothèque Prolog de base.

4. CONCLUSION

ASH-PROLOG a d'ores et déjà été utilisé dans notre laboratoire pour réaliser:

- un prototype d'un sytème explicatif destiné aux utilisateurs d'un environnement bureautique intégré;
- un outil d'aide à la mise au point de grammaires formelles en Prolog (voir Fig 2);
- un système de visualisation de graphes complexes;
- un tableau de bord pour Prolog et un tableau de bord pour un générateur de plans (en cours).

Ces expériences nous ont convaincus que Prolog constitue un langage satisfaisant pour la spécification de prototypes, et qu'il serait très souhaitable de disposer des fonctionnalités offertes par ASH-PROLOG dans un environnement disposant d'un compilateur. Nous avons également remarqué que le mécanisme de modularité offert par les prédicats consult et reconsult est un peu trop primitif (ce qui est bien connu de tous les utilisateurs habitués à ce langage).

Sous réserve d'une évolution favorable dans ces deux domaines, il serait possible d'envisager le développement de gros logiciels d'I.A. en Prolog (plusieurs milliers de clauses), les extensions graphiques présentées ici rendant possible la gestion directe de l'interaction homme-machine, et en particulier la représentation externe des états internes du système d'inférence et de la base de connaissance qui lui est associée.

BIBLIOGRAPHIE

Bratko I.; *Prolog Programming for Artificial Intelligence,* Addison-Wesley 1986.

Chailloux J.; *Le_Lisp, Le Manuel de Référence,* I.N.R.I.A., Rocquencourt 1985.

Clark I.A.; *Software simulation as a tool for usable product design,* IBM Syst. J., 20, 3, 1981.

Clocksin W.F., Mellish C.S.; *Programming in Prolog,* Springer-Verlag 1981.

Hayes P.J.; "Executable Interface Definitions Using Form-Based Interface Abstractions" in *Advances in Human-Computer Interaction,* Hartson H.R. Ed., Ablex Publishing, 1985.

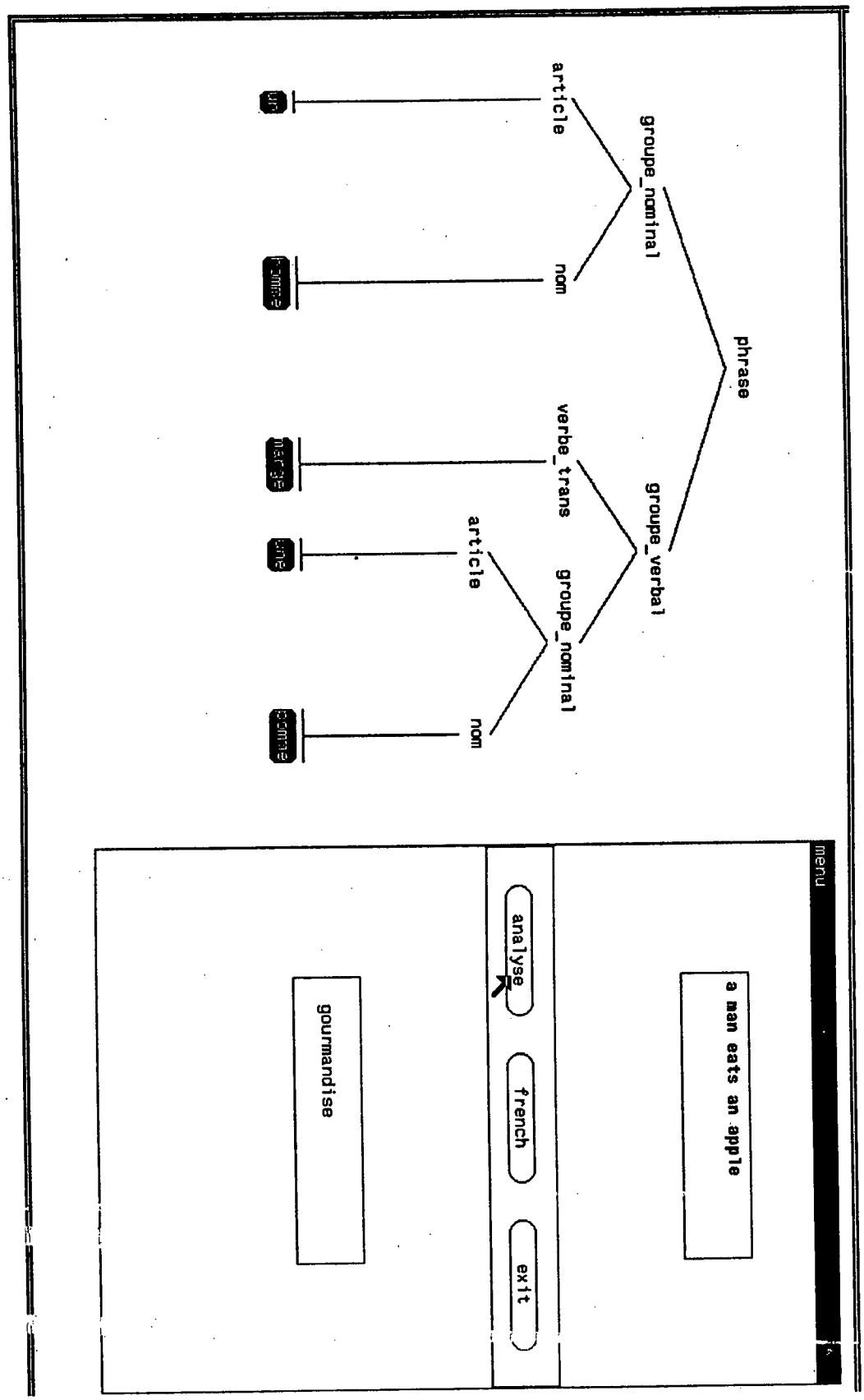
Kluzniak ; *Prolog for Programmers,* Academic Press 1985.

Pato J.N., Reiss S.P., Brown M.H.; *The Brown Workstation Environment,* Brown University, Dept. of Comp. Science, Providence 1984.

Rueher M., Legeard B., Traversat B.; *Une approche intelligence artificielle en prototypage,* Bulletin BIGRE, 43, 1985.

Wasserman A.I., Shewmake D.T.; "The Role of Prototypes in the User Software Engineering (USE) Methodology" in *Advances in Human-Computer Interaction,* Hartson H.R. Ed., Ablex Publishing, 1985.

Figure 2: visualisation du fonctionnement d'une grammaire formelle



ANNEXE : PREDICATS ASH-PROLOG (Version 2.2)

5. INITIALISATIONS

ashinit(Mode).

Initialisation de l'environnement graphique, chargement d'un curseur par défaut. *Mode* peut prendre les valeurs:

ash_mode_display
pour initialisation du plein écran;

ash_mode_window
pour initialisation d'une fenêtre graphique dans un contexte multi-fenêtres (Suntools par exemple);

ash_mode_new_window
pour initialisation d'une *nouvelle* fenêtre graphique de base.

ashinit.

Initialisation dans le mode par défaut *ash_mode_window*.

apiinit.

Initialisation des fonctions "apio" utilisées pour le traitement des "évènements utilisateurs". (Cf. infra).

ashcombination_rule(N).

Permet de modifier la règle de combinaison utilisée pour déterminer la valeur d'un pixel en cas de superposition de tracés.

6. CREATION ET MANIPULATION DE FENETRES

6.1. Création et destruction.

makewindow(Name,Parent_lx,Parent_by,Lx,By,Rx,Ty,Border_id,Flag_id).

Permet de créer une fenêtre qui devient la fenêtre courante. Les paramètres sont:

Name est le nom de la fenêtre;

Parent_lx et *Parent_by*
sont les coordonnées du coin inférieur gauche,[8] dans le repère de coordonnées de la fenêtre mère;

Lx et *By*
sont les coordonnées de ce même point dans le système de la nouvelle fenêtre;

Rx et *Ty*
sont les coordonnées du coin supérieur droit dans ce même système.[9]

Border_id
fixe définitivement le type de bordure de la nouvelle fenêtre. Les valeurs possibles sont:

ash_border_window
qui donne un bandeau large en haut de la fenêtre;

ash_border_thin
donne un mince liseré tout autour de la fenêtre;

---

[8] Attention au système de coordonnées écran : le 0,0 est à l'angle supérieur gauche.

[9] Dans la version 2.2 il faut éviter les coordonnées négatives en raison d'une erreur non encore identifiée.

ash_border_none
pour avoir une fenêtre sans bords visibles;

ash_border_sensitive
donne un simple liseré qui se transforme en un trait plus épais lorsque la souris passe sur cette fenêtre (dernier "hit");

ash_border_tab
qui donne un bandeau large sur le quart de la longueur du haut de la fenêtre;

ash_border_tabsense
comme le précédent, mais avec en plus une inversion de la bande supérieur dans sa partie vide;

ash_border_flip
provoque une inversion de la fenêtre lorsqu'elle est désignée (dernier "hit");

ash_border_tab_only
la bordure n'a qu'une simple étiquette, sans liseré ailleurs.

Flag_id
est une combinaison de paramètres modifiables en cours d'exécution du programme. Toute combinaison "raisonnable" des valeurs suivantes séparées par *or* est admise.

ash_window_visible
la fenêtre est visible dés sa création.

ash_window_invisible
la fenêtre n'est pas visible lors de sa création;

ash_window_hit_use
la fenêtre peut recevoir des désignations;

ash_window_hit_parent
les désignations concernent la fenêtre mère;

ash_window_frame
la fenêtre est un simple cadre qui n'a aucune existence autonome mais occupe le même emplacement que sa fenêtre mère;

ash_window_courteous
la fenêtre n'aura qu'une existence temporaire. L'état de l'écran à son emplacement sera sauvegardé lors de sa création, et automatiquement restauré lors de sa destruction;

ash_window_nosave
cette fenêtre n'aura pas de représentation interne en mémoire hors écran. Elle ne pourra donc pas être rendue invisible et restaurée.

ash_window_dependant
la visibilité de la fenêtre dépend de celle de sa mère (vrai par défaut);

ash_window_independent
la visibilité de la fenêtre est indépendante de celle de sa mère;

ash_window_transparent
la fenêtre est transparente aux désignations : les désignations sont transmises à la fenêtre qui se trouve "en dessous" sur l'écran.

ash_window_fast
la fenêtre est sauvegardée en mémoire rapide;

ash_window_noclear
les graphiques contenus dans la fenêtre ne peuvent être effacés. (Ceci permet de faire des effacements selectifs dans la fenêtre mère);

ashset_window_name(Label).
Permet d'inscrire dans le bandeau supérieur des fenêtres à bordure du type *ash_border_window* ou *ash_border_tab* une étiquette Label.[10]

ashremove(Name).
Détruit[11] la fenêtre de nom *Name*.

6.2. Selections et visibilité.

ashselect(Name).
La fenêtre *Name* devient la fenêtre courante pour les manipulations qui suivent.

ashvisible(X).
Si X est 0, la fenêtre n'est pas visible, si X est 1, elle l'est.

ashpush.
La fenêtre courante est "poussée au fond de l'écran" : si une autre fenêtre existe au même emplacement, cette dernière devient exposée aux désignations et visible.

ashpop.
La fenêtre courante est ramenée "au dessus de l'écran". Elle masque donc une éventuelle fenêtre qui se serait trouvée au même emplacement.

ashpush_window.
Sauve la fenêtre courante dans une pile (indépendante de la notion d'ecran).

ashpop_window.
Restaure la dernière fenêtre sauvée dans la pile.

6.3. Récupération d'informations sur les fenêtres.

ashinq_window_name(Text).
Retourne l'étiquette associée à la fenêtre courante.

ashinq_window(Name).
Retourne le nom de la fenêtre courante.

ashinq_parent(Name).
Retourne le nom de la fenêtre mère de la fenêtre courante.

ashinq_top(Window).
Retourne 'e nom de la fenêtre écran (racine de l'arbre des fenêtres).

ashinq_under(On_window,X,Y,Under_window).
Retourne le nom de la fenêtre directement au-dessous de la première *(On_window)* aux coordonnées X et Y.

ashfind_window(X,Y,Window).
Retourne le nom de la fenêtre "en sommet d'écran" aux coordonées X et Y.

ashinq_rectangle(X,Y,Lx,By,Rx,Ty,Window).
Retourne le nom et les coordonnées des angles de la fenêtre placée aux coordonnées X et Y de l'écran.

ashinq_region_visible(Window,Lx,By,Rx,Ty).
Réussit ou échoue selon que la région donnée est ou non totalement visible.

ashinq_size(Type,Lx,By,Rx,Ty).
Est utilisé pour obtenir des informations sur les différentes zones associées à la fenêtre courante. *Type* décrit de quelle zone il s'agit. En retour on obtient le coin inférieur

---

[10] Attention, cette etiquette est totalement indépendante du nom symbolique de votre fenêtre qui vous sert à la désigner tout au long de votre programme (paramétre *Name* du prédicat makewindow).

[11] Attention : c'est une destruction définitive de la fenêtre. Donc à ne pas utiliser pour rendre une fenêtre temporairement invisible.

gauche (Lx,By) et le coin supérieur droit de la zone. *Type* peut prendre les valeurs suivantes:

ash_size_window
pour obtenir la taille de la fenêtre hors bordures. Les coordonnées obtenues sont exprimées dans le système de coordonnées de la *fenêtre*.

ash_size_view
pour obtenir la taille de la zone théoriquement visible : la fenêtre ayant pu être clippée par programme, (prédicat *ashview* par exemple),cette taille n'est pas forcément équivalente à la précédente. Les coordonnées obtenues sont exprimées dans le système de coordonnées de la *fenêtre*.

ash_size_screenview
pour obtenir la taille de la zone réellement visible à l'écran. L'utilisateur a pu déplacer cette fenêtre et la positionner de telle sorte qu'une partie seulement soit visible. Cette taille n'est donc pas forcément équivalente aux deux précédentes. Les coordonnées obtenues sont exprimées dans le système de coordonnées de la *fenêtre*.

ash_size_screen
pour obtenir les coordonnées écran de la zone visible.

ash_size_border
pour obtenir la taille de la bordure en coordonnées écran.

ash_size_screen_full

ash_size_view_full

ash_size_window_full
les trois derniers retournent les coordonnées en y incluant la bordure s'il y en a une.

ashinq_border_size(Id,Left,Bottom,Right,Top).
Retourne la taille de la bordure et son type *Id*

6.4. Déplacements.

ashview(Parent_lx,Parent_ly,Lx,By,Rx,Ty).
Permet de modifier la représentation à l'écran de la fenêtre courante. En modifiant *Parent_lx* et *Parent_ly* qui sont les coordonnées du coin inférieur gauche de la fenêtre courante dans le système de coordonnées de sa mère, on peut bouger la fenêtre courante. *Lx* , *By* et *Rx* , *Ty* sont respectivement les coordonnées dans le système de la fenêtre courante des points inférieur droit et supérieur gauche de la zone rectangulaire que l'on veut voir à l'écran; il est à noter que cette primitive ne permet pas d'agrandir la fenêtre courante: en bon franglais, ce n'est pas un zoom mais un clip.

ashresize(Lx,By,Rx,Ty).
Redimensionne la fenêtre courante, la bitmap associée est détruite et la fenêtre est effacée.

ashpar_resize(Parent_lx,Parent_by,Lx,By,Rx,Ty).
A les mêmes propiétés que *ashresize,* et permet en plus de replacer la fenêtre courante dans sa fenêtre mère.

move_window(Nx,Ny).
Permet à l'utilisateur de deplacer une fenêtre en la désignant avec la souris et en déplacant celle-ci avec le bouton maintenu pressé. Un cadre de même dimensions que la fenêtre permet pendant le déplacement de voir ou elle se trouvera au relachement du bouton. Ce prédicat s'utilise toujours précédé d'un *apiogetclick* (voir rubrique "Evénements utilisateur").

6.5. Correspondance entre systèmes de coordonnées.

ashmap(From_window,From_x,From_y,To_window,To_x,To_y).

Assure la transformations des coordonnées *From_x* et *From_y* de la fenêtre *From_window* dans le repère de la fenêtre *To_window*. Les valeurs sont retournées dans *To_x* et *To_y*

ashhit(X,Y,Wx,Wy,Result).

Cherche la première fenêtre sur l'écran aux coordonnées X et Y (repère écran). *Wx* et *Wy* sont instanciées aux coordonnées correspondantes dans le repère de la fenêtre et *Result* est instancié au nom de cette dernière.

6.6. Divers

ashpush_state et ashpop_state

Permettent de sauver et de restaurer *l'état* de la fenêtre courante (et non la fenêtre elle-même).

7. PRIMITIVES GRAPHIQUES

ashpoint(X,Y).

Dessine un point aux coordonnées exprimées dans le système de la fenêtre courante.

ashpolypoint(Nb,L).

Dessine *Nb* points (X1,Y1) ... (XNb,YNb) donnés dans *L* sous la forme de la liste Prolog [X1,Y1,X2,Y2, ... XNb,YNb].

ashline(X0,Y0,X1,Y1).

Trace une droite du point X0,Y0 au point X1,Y1 les coordonnées étant décrites dans le système de la fenêtre courante.

ashpolyline(Nb,L).

Dessine une ligne brisée entre *Nb* points (X1,Y1) ... (XNb,YNb) donnés dans *L* sous la forme [X1,Y1,X2,Y2, ... XNb,YNb].

ashrectangle(X1,Y1,X2,Y2).

Dessine un rectangle plein décrit par deux de ses coins opposés.

> **ATTENTION : Pour ce prédicat et les trois qui suivent, les points descriptifs sont les angles supérieur gauche et inférieur droit. Ce n'est donc pas la même convention que pour la description des fenêtres ou des zones sensibles. L'expérience a montré que c'est là une source d'erreurs fréquentes.[12]**

ashbox(X1,Y1,X2,Y2).

Dessine un rectangle vide décrit par deux de ses coins opposés (inférieur gauche et supérieur droit).

ashround_rectangle(X1,Y1,X2,Y2,R).

Dessine un rectangle plein décrit par deux de ses coins opposés et aux bords arrondis, de rayon R (inférieur gauche et supérieur droit).

ashround_box(X1,Y1,X2,Y2,R).

Dessine un rectangle vide décrit par deux de ses coins opposés et aux bords arrondis, de rayon R.

---

[12] Nous avons conservé cette caractéristique assez pénible au début, dans le soucis de préserver la cohérence avec ASH et avec ASH-Le_lisp : il nous a semblé que les programmeurs C ou Lisp souhaiteraient conserver leurs habitudes en découvrant ASH-PROLOG.

ashpolygon(Nb,Liste) , ashconvex_polygon(Nb,Liste).

Dessinent un polygone plein convexe avec *Nb* points (X1,Y1) ... (XNb,YNb) donnés dans *Liste* sous la forme [X1,Y1,X2,Y2, ... XNb,YNb].

ashgeneral_polygon(Nb,Liste).

Dessine un polygone plein concave avec *Nb* points (X1,Y1) ... (XNb,YNb) donnés dans *Liste* sous la forme [X1,Y1,X2,Y2, ... XNb,YNb].

ashcircle(X,Y,R).

Dessine un cercle de rayon *R* aux coordonnées de centre *X* et *Y* de la fenêtre courante.

ashfilled_circle(X,Y,R).

Dessine un cercle plein de rayon *R* aux coordonnées *X* et *Y* de la fenêtre courante.

ashclear.

Efface les graphiques dans la fenêtre courante.

ashcopy_drawinfo(Name).

Transfère les informations graphiques de la fenêtre *Name* vers la fenêtre courante.

ashpush_drawinfo , ashpop_drawinfo.

Permettent de sauver et de restaurer l'état graphique de la fenêtre courante.

ashfill(Pattern_id).

Fixe le mode de remplissage des figures pleines de la fenêtre courante. Les différentes valeurs de *Pattern_id* sont *ash_fill_pattern_0* à *ash_fill_pattern_15*. Les résultats dépendent de ce qu'autorise la bibliothèque "raster-op" de la machine hôte.

ashline_style(Pattern_id).

Fixe le style de ligne des graphiques dans la fenêtre courante. Les différentes valeurs possibles de *Pattern_id* sont :

ash_line_hollow

ash_line_solid

ash_line_pattern

ash_line_hatch

Les résultats dépendent de ce qu'autorise la bibliothèque "raster-op" de la machine hôte.

ashinq_fill(Id), ashinq_line_style(Id).

Renvoient les valeurs utilisées par la fenêtre courante.

ashclip_region(Lx,By,Rx,Ty).

Définit la région dans laquelle il sera possible d'effectuer des dessins dans la fenêtre courante.

ashclip(Flag).

Active (*Flag* = 1) ou désactive (*Flag* = 0) les instructions données dans ashclip_region.

8. ECRITURE DE CARACTERES

ashtext(X,Y,T).

Ecrit le texte *T* aux coordonnées *X*, Y de la fenêtre courante. *T* peut être n'importe quel arbre Prolog. Si c'est un atome, on obtient son nom, si c'est un arbre, sa représentation Prolog. Exemples:

Prédicat	Resultat
ashtext(X,Y,foo).	foo
ashtext(X,Y,'FOO').	FOO
ashtext(X,Y,foo(bar(X),truc(Z))).	foo(bar(X),truc(Z))

ashcenter_text(Text,LX,BY,RX,TY).

Ecrit le texte *Text* centré dans le rectangle décrit. Le rectangle est virtuel: il n'est pas tracé. Si l'on souhaite encadrer ce texte en utilisant *ashbox* par exemple, il est

souhaitable de faire d'abord le *ashcenter_text* puis le *ashbox* et non l'inverse.

ashfont(Font).

Permet de choisir la police de caractères qui sera utilisée pour les écritures par *ashtext, ashcenter_text,* etc.. *Font* peut être soit une adresse absolue commencant par un / , soit une directement le nom du fichier contenant la police. Dans ce cas une règle de recherche spécifique est utilisée qui comprend par défaut votre "home directory", et le répertoire standard des polices dont vous disposes sur votre station de travail (par défaut le repertoire /usr/lib/fonts/fixedwidthfonts). Vous pouves rajouter d'autres répertoires à la regle de recherche utilisée en utilisant le prédicat

asserta(directory(font,Path)).

dans lequel *Path* est instancié au pathname à rajouter.

ashinq_font(Name).

Renvoie le nom de la police courante.

ashinq_text_extent(String,X,Y).

ashinq_text_offset(String,X,Y).

ashinq_text_next(String,X,Y).

Ces trois prédicats retournent des informations sur la taille et l'espacement des chaînes de caractères à écrire dans la fenêtre courante avec la police sélectionnée. Le premier donne la longueur et la hauteur de la chaîne. Le deuxième est utilisé pour écrire dans un rectangle: il retourne les distances entre le coin inférieur gauche et l'endroit où serait imprimé le texte avec ashtext. Le troisième retourne des informations pour ajouter une nouvelle chaîne : *X* est l'emplacement pour concatèner le texte, *Y* est la différence à respecter pour écrire sur la ligne suivante.

vt_open.

Initialise la fenêtre courante en terminal virtuel. Les graphiques et textes qui se trouveraient déjà dans la fenêtre sont effacés.

vt_display(String).

Ecrit la chaine de caractères à la position courante du curseur virtuel de la fenêtre utilisée en terminal. *String* doit apparaitre entre simples quotes. Elle peut comprendre des caractères de contrôle de déplacement du curseur, de positionnement d'attributs vidéo, etc.. La description de ces séquences de contrôle est fournie dans la documentation du "Brown Workstation Environment" dont nous reproduisons les quelques pages pertinentes en Annexe2. Il est de bon usage d'utiliser *vt_display* pour *chaque* ligne à écrire, la *String* étant limitée à 256 caractères. Nous renvoyons à l'exemple développpé dans la première partie du présent manuel, pour illustrer une utilisation de vt_display pour l'affichage de fichiers dans une fenêtre.

vt_close.

Fermeture du terminal virtuel de la fenêtre courante.

9. ZONES SENSIBLES

makesensitive_area(Name,Lx,By,Rx,Ty,Flag_id).

Permet de créer une zone sensible aux désignations dans la fenêtre courante. *Name* est le nom de cette zone sensible, *Lx, By* sont les coordonnées de son angle inférieur gauche, *Rx, Ty* sont celles de l'angle supérieur droit. *Flag_id* peut prendre les valeurs suivantes:

ash_sense_flip

La zone sensible passera en inverse vidéo chaque que le curseur passera dessus;

ash_sense_no_change

Pas de modification vidéo lors du passage du curseur.

ashsensitive_remove(Name).

La zone sensible *Name* est détruite.

ashsensitive_remove_all.

Toutes les zones sensibles de la fenêtre courante sont détruites.

ashinq_sensitive(Name,Switch).

Attend qu'une désignation d'une zone sensible soit faite, et retourne le nom de celle-ci et le numero du bouton de la souris utilisé, dans le cas d'une souris à plusieurs boutons.[13]

ashhit_window(Window).

"Simule" une désignation de la fenêtre *Window*. La seule (?) utilité de ce prédicat et de forcer un surlignage vidéo des bordures du type *ASH_BORDER_SENSITIVE*. (cf les types de bordures dans *makewindow*).

ashhitable(hittype).

Modifie la sensibilité de la fenêtre courante aux désignations. Le paramètre *hittype* peur prendre les valeurs suivantes :

ash_hit_ignore

La fenêtre ignore les désignations.

ash_hit_use

La fenêtre accepte les désignations.

ash_hit_parent

Les désignations sont transmises à la fenêtre mère.

ash_hit_transparent

La fenêtre est transparente aux désignations, qui touchent donc la fenêtre se trouvant "en dessous" sur l'écran.

10. EVENNEMENTS UTILISATEUR

apiogetchar(X).

Attend la frappe d'un caractère au clavier. Rend ce caractère.

apiogetstring(String).

Bloque l'exécution du programme et attend la frappe d'une chaîne de caractères terminée par un retour chariot. Assure l'echo des caractères frappés dans la fenêtre courante, et permet à l'utilisateur de corriger sa frappe avec le caractère <retour>. La chaîne se trouvera dans *String*

apiogetclick(X,Y,S).

Attend un "click" souris, et rend les coordonnées écran et le bouton utilisé.

apioflush.

Vide la pile des évènements clavier et souris.

ashpopup(Name).

Rend le nom de la zone sensible qui était désignée au moment du *relachement* du bouton de la souris. S'emploie (précédé d'un *apiogetclick*) pour récupérer des désignations dans les menus dynamiques.

ashinq_sensitive(Name,Switch).

Attend qu'une désignation d'une zone sensible soit faite, et retourne le nom de celle-ci et le numero du bouton de la souris utilisé, dans le cas d'une souris à plusieurs boutons.

interrupt(Action).

Spécifie le traitement à effectuer en cas d'appui de la touche <DEL> pendant une attente sur *ashinq_sensitive, apiogetchar, apiogetclick ou apiogetstring.*
Action peut prendre les valeurs suivantes:

fail Echec du prédicat d'attente.

---

[13] Sur Sun, *Switch* peut prendre les valeurs 1, 2 ou 4.

abort Arrêt du programme. (Identique à l'évaluation du prédicat Prolog *abort*.

exit Arrêt de Prolog et retour à Unix.

11. MANIPULATION DU CURSEUR

ashcursor(X).

Si X est 0 le curseur n'est pas visible, et si X est 1 le curseur redevient visible.

ashcursor_restore.

Remplace le curseur courant par le curseur original.

ashcursor_load(Type).

Le fichier cursors.font propose 50 types de curseurs prédéfinis. Leurs noms sont:

ash_cursor_arrow_ur	*ash_cursor_arrow_ll*
ash_cursor_arrow_ul	*ash_cursor_arrow_lr*
ash_cursor_glasses	*ash_cursor_bullseye*
ash_cursor_xhairs	*ash_cursor_small_glasses*
ash_cursor_small_bullseye	*ash_cursor_small_xhairs*
ash_cursor_x	*ash_cursor_clown*
ash_cursor_dot	*ash_cursor_cross*
ash_cursor_small_x	*ash_cursor_star*
ash_cursor_circle	*ash_cursor_square*
ash_cursor_box_x	*ash_cursor_focus*
ash_cursor_pinwheel	*ash_cursor_diamond*
ash_cursor_snowflake	*ash_cursor_asterisk*
ash_cursor_pipe	*ash_cursor_circle_box*
ash_cursor_square_x	*ash_cursor_mouse*
ash_cursor_run	*ash_cursor_stop*
ash_cursor_character	*ash_cursor_character_std*
ash_cursor_character_std_19l	*ash_cursor_watch*
ash_cursor_hourglass	*ash_cursor_between_l*
ash_cursor_between_r	*ash_cursor_between_small_l*
ash_cursor_between_small_r	*ash_cursor_tying*
ash_cursor_xxx_0	*ash_cursor_xxx_1*
ash_cursor_xxx_2	*ash_cursor_xxx_3*
ash_cursor_xxx_4	*ash_cursor_xxx_5*
ash_cursor_xxx_6	*ash_cursor_xxx_7*
ash_cursor_xxx_8	*ash_cursor_xxx_9*

Il est possible d'examiner cette police de curseurs, et d'en créer de nouveaux, avec un éditeur de police de caractères.

ashcursor_move(X,Y).

Déplace le curseur à la position X, Y.[14]

ashpush_cursor(Flag), ashpop_cursor.

Sauve et rappelle le curseur courant. *Flag* (0 ou 1) indique si le curseur reste visible aprés le push.

ashinq_cursor(Id).

Rend le nom du curseur courant.

ashcursor_define(Id,Ch,Font,X,Y,Xor_flag).

Permet à l'utilisateur de définir un nouveau curseur, préalablement dessiné dans une police à l'aide d'un éditeur de polices de caractères. *Id* identifie le curseur qui sera défini. Dix identificateurs sont prévus pour cette création (*ash_cursor_user_0* à *ash_cursor_user_9*

[14] Cette fonction est inopérante sur certains postes de travail (Sun par ex), le "tracking" de la souris par le curseur étant pris en charge à un niveau très bas.

). *Ch* et *Font* identifient le caractère et la police utilisés. *X* et *Y* définissent le décalage du point repère par rapport au coin supérieur gauche. Enfin *Xor_flag* indique si la règle de combinaison sera OR (false) ou XOR (true).

12. MANIPULATIONS DE "BIT-MAPS"

ashsave_bitmap(File), ashload_bitmap(File).

Sauvegarde et rappelle une bitmap de la fenêtre courante dans le fichier *File*.

ashload_icon(X,Y,Name).

Place en X, Y de la fenêtre courante une icone contenue dans le fichier *Name*. Cette icone a été définie à l'aide de l'editeur d'icones disponible sur votre système. *Name* peut être une adresse absolue (commencant par /), ou un nom de fichier qui sera recherché dans le repèrtoire de travail, et dans /usr/local/icons.[15]

ashblt(Lx,By,Rx,Ty,Dlx,Dby).

copie la région de la fenêtre source déterminée par les quatre premières coordonnées dans la fenêtre courante en positionnant le coin inférieur droit aux coordonnées *Dlx*, Dby.

ashsource(Window).

Spécifie que la fenêtre *Window* doit être prise comme source pour ashblt.

13. DIVERS

ashcombination_rule(X).

Etablit les règles de superposition pour la fenêtre courante (AND,OR,XOR,...). *X* est compris entre 0 et 15. Le tableau ci-après donne la valeur du pixel résultat en cas de superposition.

Source	Destination	Resultat selon la règle de combinaison															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	1	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

ashinq_combination_rule(X).

Rend le numéro de la règle de combinaison courante.

ashnew_frame.

Rafraichit l'écran.

interrupt(X).

Définit l'effet de la touche < DEL> . Voir Chapitre "Evénements Utilisateur".

ashset_userdata(Foo).

Permet d'associer à la fenêtre courante une information quelconque, d'une facon plus efficace que par l'utilisation d'un *assert*. *Foo* peut être n'importe quel atome ou arbre Prolog mais ne peut plus être modifié pour une fenêtre donnée.

ashinq_userdata(F).

Permet de récupérer l'information associée à la fenêtre courante.

---

[15] Dans la version 2.2, cette fonction n'est disponible que sur Sun.

ANNEXE 2 : SEQUENCES DE CONTROLE DU TERMINAL VIRTUEL

AVERTISSEMENT : La présente documentation est un extrait de REISS S.P.; Virtual Terminal, in *THE BROWN WORKSTATION ENVIRONMENT*, Brown University, Providence, 1983.

Control Characters

Text strings passed to the virtual terminal may contain a limited set of control characters that have special meanings. All other control characters will be ignored. The relevant characters are:

^j LINE-FEED sets the current position to be the start of the next line. If new-line clear mode is set, then the rest of the current line is cleared first.

^m CARRIAGE-RETURN goes to the beginning of the current line.

^h BACK-SPACE backs up one character. If backspace clear mode is set, then the character that is backed up over is erased.

^i If tab clear mode is set then TAB inserts spaces up to the next tab stop, else it just moves the current position to the next tab stop. Tab stops are every 8 characters regardless of character size. For fixed column tabs, use the SET TAB and USE TAB escape sequences.

^g BELL Displays/sounds a bell.

^@ NULL marks the end of the text string.

^[ESCAPE marks the beginning of an escape sequence.

Escape Sequences

Most of the fancy work done by the virtual terminal package is controlled using escape sequences. These are stolen from ANSI-type sequences and have one of the forms:

ESCAPE < key>
ESCAPE [< parameter> ;< parameter> ;... < key>
ESCAPE < mode> < value>

where < key> is in the range 0100-0176, parameters are non-negative integers, < mode> is in the range 040-077, and value is a single character. Any control character in the middle of an escape sequence will cause it to be ignored. Similarly, any invalid escape sequence will be ignored.

Initially, most escape sequences are one of the < key> forms. Where parameters are optional, reasonable defaults are assumed. The following is an initial layout of the keys and their interpretation:

A Move up by the given number (1) of lines.

B Move down by the given number (1) of lines.

b Set the text background color to the value specified by the first parameter.

C Move right by the given number (1) of characters (with spaces of the current font assumed beyond the end of line).

c Set the text color to the color specified by the first parameter.

D Move left by the given number (1) of characters, stopping at column 0.

E Redraw the window.

F Change font to the one specified by the parameter (standard).

G Define a graphics region at this point whose size is given in pixels by the two parameters. (Not implemented yet.)

H Position the cursor at the line,column location given by the parameters (home).

h Position the cursor at the line and column corresponding to the location given in pixel coordinates by the parameters (home).

I Enter insert mode.

i Exit insert mode.

J Erase within the display. If the parameter is 0 (or is not given) then erase from current position to the end of the terminal. If the parameter is 1 then erase from the start of the display to the current position. If the parameter is 2 then erase the whole display.

K Erase within the line. If the parameter is 0 (or if none is given) then erase from the current position to the end of the line. If the parameter is 1 then erase from the start of the line to the current position. If the parameter is 2 then erase the whole line.

L Define rectangular marked region. A parameter of 1 indicates the starting point of the region. A parameter of 0 (or no parameter) indicates a new end point. A parameter of 2 terminates this mode. All data in the given region will be displayed in inverse video.

l Set the line limit -- the maximum number of lines allowed by the virtual terminal. Once this limit has been reached, additional lines will cause the earlier lines to be discarded.

M Define a textual marked region. This is similar to L above except that the region is character oriented -- contains all the text from the starting point to the ending point and not just the rectangle defined by those points.

N Delete the given number (1) of lines starting at the current line.

O Delete the given number (1) of characters starting at the current character.

o Report the origin of the current virtual terminal in pixels. A string of the form 'line column0 becomes available through VTread after this string is output.

P Insert the given number (1) of lines in front of current line.

p Split the current line at the current character position.

Q Insert the given number (1) of characters in front of current character.

q Unsplit (merge) the current line with the next line, starting the next line at the current character position.

R The two parameters define the origin of the part of the virtual terminal that will be displayed. The default is (0,0).

S A tab stop is set at the current position. The parameter serves as the name of the tab stop. It is possible to line up future columns to this tab°stop using escape-T. Currently 10 tab stops (0-9) are provided.

s The size of the current virtual terminal is placed in a buffer to be read by a subsequent call to VTread. The size is the estimated number of lines and columns (in that order) that can be placed in the window using the current font.

T Tab to the tab stop indicated by the parameter. This goes to the closest character in the current line.

t The x position of the tab specified by the first parameter is returned thru the next call to VTread.

U Draw a rectangular box around the area of text. The first parameter identifies the box. If it is the only parameter supplied, then the box is removed. Four additional parameters are used to define the box -- the starting line and column and the ending line and column in that order.

u This is similar to U except that a filled rather than rectangular box is drawn.

V Return the number of characters (through VTread) of the given (current) line.

W Map all text in the region specified by the first four parameters to a different font. The parameters after the first four are source font - destination font pairs that define the transformation to be used.

w The current cursor position (line column0 is returned through a subsequent call to VTread.

X Draw a text box around the area of text indicated by the four parameters. This command is similar to U except that the box encompasses all text, not just a rectangle.

x This is similar to X except that a filled rather than rectangular box is drawn.

Z Enable or disable (default) the display of the cursor. If the parameter is 1 then the cursor will be displayed. If it is 0 then no cursor will be displayed. The cursor is displayed at the current position as long as that is in the displayed window.

z The current character and its font are returned through a subsequent call to VTread. Two characters are output. The first is the actual displayed character. The second represents the font. This is the character '@' plus the font number (i.e. font 0 is @, font 1 is A, ...).

Parameters

The mode-value form of escape sequence is used for parameter settings. The following parameters are user-settable:

Scrolling

The escape sequence <ESCAPE>!<digit> is used to set the scrolling options. Any reasonable combination (OR) of the following is allowed: 1 -- current position mode -- the area displayed will be moved so that the current position is always displayed; 2 -- truncate mode -- only information that fits on the screen will be displayed, the rest will be lost (not implemented yet).

Optional clearing

The escape sequence <ESCAPE>#<digit> is used to tell the VT package when to do automatic clearing. ESCAPE#0 resets and ESCAPE#1 sets a mode whereby a newline character clears up to the end of line. ESCAPE#2 resets and ESCAPE#3 sets the mode whereby a backspace clears the character being backspaced over. ESCAPE#4 resets and ESCAPE#5 sets the mode whereby a tab clears the characters being tabbed over.

Frame mode

The escape sequence <ESCAPE>(1 will cause VT to limit its display manipulations to those that are valid in a frame. <ESCAPE>(0 resets this mode.

Absolute mode

The escape sequence <ESCAPE>)1 will cause VT to enter absolute mode. In absolute mode moving up or down lines will not change the character position. <ESCAPE>)0 resets this mode. In the reset mode, moving up or down lines moves to the closest character in the new line, which, because of different fonts, may be at a different character position.

Fonts

The virtual terminal package is capable of handling and intermixing a variety of fonts, both fixed and variable sized. Initially each virtual terminal has only the standard font associated with it. Additional fonts may be loaded into the terminal with the sequence:

<ESCAPE>f<font_name>\n

The font name is actually terminated by a space or any control character. This terminator is otherwise ignored. After this escape sequence the virtual terminal package will place the font number in the input buffer to be read by a call to VTread. This will be a string of the form 'id\n' where id is a non-negative integer. This id should be used as the parameter for escape-F

to actually change the font. A maximum of 16 different fonts can be loaded in all of the windows. (Loading the same font in two different virtual terminals only counts as one of the sixteen.)

INDEX

PREDICATS	Page
apioflush.	23
apiogetchar(X).	23
apiogetclick(X,Y,S).	23
apiogetstring(String).	23
apioinit.	16
ashblt(Lx,By,Rx,Ty,Dlx,Dby).	25
ashbox(X1,Y1,X2,Y2).	20
ashcenter_text(Text,LX,BY,RX,TY).	21
ashcircle(X,Y,R).	21
ashclear.	21
ashclip(Flag).	21
ashclip_region(Lx,By,Rx,Ty).	21
ashcombination_rule(N).	25
ashconvex_polygon(Nb,Liste).	21
ashcopy_drawinfo(Name).	21
ashcursor(X).	24
ashcursor_define(Id,Ch,Font,X,Y,Xor_flag).	24
ashcursor_load(Type).	24
ashcursor_move(X,Y).	24
ashcursor_restore.	24
ashfill(Pattern_id).	21
ashfilled_circle(X,Y,R).	21
ashfind_window(X,Y,Window).	18
ashfont(Font).	22
ashgeneral_polygon(Nb,Liste).	21
ashhit(X,Y,Wx,Wy,Result).	20
ashhit_window(Window).	23
ashhitable(hittype).	23
ashinit(Mode).	16
ashinit.	16
ashinq_border_size(Id,Left,Bottom,Right,Top).	19
ashinq_combination_rule(X).	25
ashinq_cursor(Id).	24
ashinq_fill(Id).	21
ashinq_font(Name).	22
ashinq_line_style(Id).	21
ashinq_parent(Name).	18
ashinq_rectangle(X,Y,Lx,By,Rx,Ty,Window).	18
ashinq_region_visible(Window,Lx,By,Rx,Ty).	18
ashinq_sensitive(Name,Switch).	23
ashinq_size(Type,Lx,By,Rx,Ty).	18
ashinq_text_extent(String,X,Y).	22
ashinq_text_next(String,X,Y).	22
ashinq_text_offset(String,X,Y).	22
ashinq_top(Window).	18

ashinq_under(On_window,X,Y,Under_window).	18
ashinq_userdata(F).	25
ashinq_window(Name).	18
ashinq_window_name(Text).	18
ashline(X0,Y0,X1,Y1).	20
ashline_style(Pattern_id).	21
ashload_bitmap(File).	25
ashload_icon(X,Y,Name).	25
ashmap(From_window,From_x,From_y,To_window,To_x,To_y).	20
ashnew_frame.	25
ashpar_resize(Parent_lx,Parent_by,Lx,By,Rx,Ty).	19
ashpoint(X,Y).	20
ashpolygon(Nb,Liste).	21
ashpolyline(Nb,L).	20
ashpolypoint(Nb,L).	20
ashpop.	18
ashpop_cursor.	24
ashpop_drawinfo.	21
ashpop_state.	20
ashpop_window.	18
ashpopup(Name).	23
ashpush.	18
ashpush_cursor(Flag).	24
ashpush_drawinfo.	21
ashpush_state.	20
ashpush_window.	18
ashrectangle(X1,Y1,X2,Y2).	20
ashremove(Name).	18
ashresize(Lx,By,Rx,Ty).	19
ashround_box(X1,Y1,X2,Y2,R).	20
ashround_rectangle(X1,Y1,X2,Y2,R).	20
ashsave_bitmap(File).	25
ashselect(Name).	18
ashsensitive_remove(Name).	22
ashsensitive_remove_all.	23
ashset_userdata(Foo).	25
ashset_window_name(Label).	18
ashsource(Window).	25
ashtext(X,Y,T).	21
ashview(Parent_lx,Parent_ly,Lx,By,Rx,Ty).	19
ashvisible(X).	18
asserta(directory(font,Path)).	22
interrupt(Action).	23
makesensitive_area(Name,Lx,By,Rx,Ty,Flag_id).	22
makewindow(Name,Parent_lx,Parent_by,Lx,By,Rx,Ty,Border_id,Flag_id).	16
move_window(Nx,Ny).	19
vt_close.	22
vt_display(String).	22
vt_open.	22